```
import torch
import torchvision as tv
from rbm_mnist import RBM           # El archivo de la clase anterior.


class DBN( torch.nn.Module):
    def __init__( _, sizes, CD_k=1):
        assert( isinstance(sizes,list) or isinstance(sizes,tuple))
        assert( all(type(size)==int and size>0 for size in sizes))
        assert( len(sizes)>1)
        super().__init__()
        _.subnet = torch.nn.ModuleList()
        for i in range(len(sizes)-2):
            _.subnet.append( RBM( sizes[i], sizes[i+1], CD_k))
        _.output = torch.nn.Linear( sizes[-2], sizes[-1])

    def forward( _, v, depth=None):     # Hasta que nivel se activa.
        assert( depth==None or 0<=depth<len(_.subnet))
        vi = v
        if depth is not None:           # None es la red entera.
            for i in range(depth):
                hp, vi = _.subnet[i].sample_h( vi)
            vp, vo = _.subnet[depth].forward( vi)
        else:
            for rbm in _.subnet:
                hp, vi = rbm.sample_h( vi)
            vo = _.output( hp)
        return vi, vo


T = 20
B = 50
trn_data = tv.datasets.MNIST( root='./data',
                              train=True,
                              download=True,
                              transform=tv.transforms.ToTensor() )
tst_data = tv.datasets.MNIST( root='./data',
                              train=False,
                              download=True,
                              transform=tv.transforms.ToTensor() )

trn_load = torch.utils.data.DataLoader( dataset=trn_data,
                                        batch_size=B,
                                        shuffle=True)
tst_load = torch.utils.data.DataLoader( dataset=tst_data,
                                        batch_size=B,
                                        shuffle=False)
```

```
sizes = [ 28*28, 500, 500, 2000, 10]    # La arquitectura de Hinton.
model = DBN( sizes)

model.train()
for depth in range(len(model.subnet)):  # Entrena progresivamente los RBMs.
    rbm = model.subnet[depth]
    optim = torch.optim.SGD( rbm.parameters(), 0.01)
    print( "Depth:", depth)
    for t in range(T):
        E = 0
        for images, labels in trn_load:
            optim.zero_grad()
            data = images.view( -1, 28*28)
            v0, vk = model( data, depth)
            loss = rbm.free_energy(v0)-rbm.free_energy(vk)
            loss.backward()
            optim.step()
            E += loss.item()
        print( t, E)

optim = torch.optim.Adam( model.parameters())
costf = torch.nn.CrossEntropyLoss()
print("Model")
for t in range(T):
    E = 0
    for images, labels in trn_load:     # Con los RBMs entrenados hace el
        optim.zero_grad()               # ajuste fino con la red entera.
        data = images.view( -1, 28*28)
        x, y = model( data)
        loss = costf( y, labels)
        loss.backward()
        optim.step()
        E += loss.item()
    print( t, E)


model.eval()
r, t = 0, 0
with torch.no_grad():
    for images, labels in tst_load:
        v = images.view( -1, 28*28)
        x, y = model( v)
        r += (y.argmax(dim=1)==labels).sum().item()
        t += len(labels)
print( "Accuracy:", 100*r/t)
```